Trabajo Práctico 0: Infraestructura básica

Fabrizio Cozza, *Padrón Nro. 97.402*
fabrizio.cozza@gmail.com

Kevin Cajachuán, *Padrón Nro. 98.725*
kevincajachuan@hotmail.com

Luciano Giannotti, *Padrón Nro. 97.215*
luciano_giannotti@hotmail.com.ar

1er. Cuatrimestre de 2018
66.20 Organización de Computadoras – Práctica Martes
Facultad de Ingeniería, Universidad de Buenos Aires

1. Objetivos

Este Trabajo Práctico tiene el fin de ayudarnos a familiarizarnos con las herramientas de software que utilizaremos posteriormente en otros trabajos, como es el emulador **gxemul** para correr programas sobre una maquina MIPS con el Sitema Operativo NetBSD.

2. Programa

El software de este trabajo esta escrito en lenguaje C y permite dibujar **Julia Sets** o **Conjuntos de Julia** segun los parámetros que le pasamos por línea de comando. Estos parÃ¡metros son la region del plano complejo: delimitada por un centro, un ancho y un alto; una semilla que afectara el calculo para cáda pixel; la resolución y la salida ya sea por pantalla o por archivo. El formato a usar es PGM o *portable gray format*, que resulta útil para describir imágenes digitales en escala de grises.

3. Implementación

Una vez recibidos los parámetros, para dibujar el Julia Set el programa obtiene de cada píxel de la ventana a un punto en el plano complejo. A ese punto se lo eleva al cuadrado y le suma la semilla mencionada en la sección anterior. Esto se repite hasta que el valor absoluto del resultado sea menor a 2, en cuyo caso se toma la cantidad de iteraciones y se imprime en el archivo PGM, representando el nivel de blanco de ese piíxel.

Implementamos la clase **complex** para representar los numeros complejos.

También implementamos las siguientes funciones dentro del main.c para dibujar los conjuntos:

```
complex addComplexNumbers(complex a,complex b)
```

Suma dos números complejos.

```
complex sqrComplex(complex a)
```

Calcula la raíz cuadrada de un número complejo.

```
double absComplex(complex a)
```

Calcula el valor absoluto de un complejo.

```
int isValidNumber(char* arg)
```

Verifica que los valores ingresados para el ancho y el alto de la imagen sean válidos. De ser así devuelve 1, caso contrario devuelve 0.

```
int isValidRes(char* arg)
```

Verifica que el valor ingresado para la resolución sea válido. De ser así devuelve 1, caso contrario devuelve 0.

```
int isValidComplex(char* arg)
```

Verifica que el valor ingresado para el centro sea válido. De ser así devuelve 1, caso contrario devuelve 0.

```
int processImage(int resW, int resH,
                 complex pPos, complex seed,
                 double w, double h,
                 FILE* im, int N)
```

Contiene la lógica del dibujo de los sets. Recibe el alto y ancho de la misma, el centro, la semilla, la resolución y la cantidad de iteraciones máximas. Cada píxel dentro de la resolucíon ingresada es uno por uno transformado a números complejos teniendo en cuenta el ancho y alto y el centro y se procesa el nivel en la escala de grises correspondiente a ese píxel como se explica al principio de la sección. Estos niveles se escriben en un buffer y luego en un archivo siguiendo el formato PGM. La función retorna -1 si ocurrió algún error.

El programa se compila con el siguiente comando:

```
$gcc main.c -o tp0 -lm -std=gnu99
```

4. Código C

En esta sección colocaremos el código fuente del programa en lenguaje C.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define DEFAULT_WIDTH_RES 640;
#define DEFAULT_HEIGHT_RES 480;
#define DEFAULT_REALIMAGINARY 0;
#define DEFAULT_WIDTH_HEIGHT 2;
#define DEFAULT_REALSEED -0.726895347709114071439;
#define DEFAULT_IMAGINARYSEED 0.188887129043845954792;

typedef struct{
        double x,y;
}complex;

complex addComplexNumbers(complex a,complex b){
        complex c;
        c.x = a.x + b.x;
        c.y = a.y + b.y;
        return c;
}

complex sqrComplex(complex a){
        complex c;
        c.x = a.x*a.x - a.y*a.y;
        c.y = 2*a.x*a.y;
        return c;
}
```

```
double absComplex(complex a){
        return sqrt(a.x*a.x + a.y*a.y);
}

int processImage(int resW, int resH,
                 complex pPos, complex seed,
                 double w, double h,
                 FILE* im, int N){

        int x,y,i;
  complex z0,z1;

    if(fprintf(im, "P2 \n") < 0){
        perror("Error impreso por perror");
        return -1;
    }
    if(fprintf(im, "%d %d \n",resW,resH) < 0){
        perror("Error impreso por perror");
        return -1;
    }
    if(fprintf(im, "%d \n", N) < 0){
        perror("Error impreso por perror");
        return -1;
    }

  for(y=0;y<resH;y++){
        for(x=0;x<resW;x++){
                        // Valor z segun posicion del pixel
      z1.x=pPos.x-w/2+w/(double)resW/2+w/(double)resW*x;
      z1.y=pPos.y+h/2-h/(double)resH/2-h/(double)resH*y;
      z0.x = 0;
      z0.y = 0;
      for(i=0;i<N-1;i++){
                                z0 = addComplexNumbers(sqrComplex(z1),seed);
        z1=z0;
        if (absComplex(z0) > 2.0){
                break;
        }
        i++;
        }
      //agregar al buffer el brillo
      if(fprintf(im, "%3d ", i) < 0){
            perror("Error impreso por perror");
            return -1;
      }
    }
                if(fprintf(im, "\n") < 0){
             perror("Error impreso por perror");
             return -1;
                }
        }

        /* close the file */
        return fclose(im);
}

int isValidNumber(char* arg) {
        for(int i = 0; arg[i] != '\0'; i++) {
        // CERO Y NUEVE RESPECTIVAMENTE
                if((arg[i] < 48) || (arg[i] > 57)) {
                        return 0;
                }
        }
        return 1;
}

int isValidRes(char* arg) {
        for(int i = 0; arg[i] != '\0'; i++) {
                if(arg[i] < '0' || arg[i] > '9') {
                        if(arg[i] == 'x') {
                                if(i == 0 || arg[i+1] == '\0') return 0;
                                continue;
                        }
                        return 0;
```

```
                }
        }
        return 1;
}

int isValidComplex(char* arg) {
        for(int i = 0; arg[i] != '\0'; i++) {
                if(arg[i] < '0' || arg[i] > '9') {
                        if(arg[i] == 'i') {
                                if(arg[i+1] != '\0') return 0;
                                continue;
                        }

                        else if(arg[i] == '+') {
                                if(arg[i+1] == '\0' || arg[i+1] == '-' || arg
                                     [i+1] == 'i') return 0;
                                continue;
                        }

                        else if(arg[i] == '-') {
                                if(arg[i+1] == '\0' || arg[i+1] == '+' || arg
                                     [i+1] == 'i') return 0;
                                continue;
                        }
                        return 0;
                }
        }
        return 1;
}

int main(int argc, char* argv[]){
        int exitCode = 0;
        int pasoN = 500;

  int resWidth;
  int resHeight;
  complex pixelPos;
  double width;
  double height;
  complex seed;
        FILE* image = stdout;

  const char* delRes = "x";
        const char* delimitator = "+-i";
        char *pSeparator;

  resWidth = DEFAULT_WIDTH_RES;
  resHeight = DEFAULT_HEIGHT_RES;
  pixelPos.x = DEFAULT_REALIMAGINARY;
  pixelPos.y = DEFAULT_REALIMAGINARY;
  width = DEFAULT_WIDTH_HEIGHT;
  height = DEFAULT_WIDTH_HEIGHT;
  seed.x = DEFAULT_REALSEED;
  seed.y = DEFAULT_IMAGINARYSEED;

  for (int i = 1; i < argc; ++i){
        if (((!strcmp(argv[i],"-V"))) || ((!strcmp(argv[i],"--version")))){
        printf("TP0 Organizacion de Computadoras version \"1.0.0\"\
              \n\nIntegrantes:\n Fabrizio Cozza\n Kevin CajachuÃ¡n\n Luciano
                  Giannotti\n");
      return 0;
                }
    else if (((!strcmp(argv[i],"-h"))) || ((!strcmp(argv[i],"--help")))){
      printf("\
Uso:\n\
  tp0 -h\n\
  tp0 -V\n\
  tp0 [options]\n\
Opciones:\n\
  -V, --version     Version del programa.\n\
  -h, --help        Informacion acerca de los comandos.\n\
  -r, --resolution  Cambiar resolucion de la imagen.\n\
  -c, --center      Coordenadas correspondientes al punto central.\n\
```

```
  -w, --width       Especifica el ancho de la region del plano complejo por
      dibujar.\n\
  -H, --height      Especifica el alto de la region del plano complejo por
      dibujar.\n\
  -s, --seed        Configurar el valor complejo de la semilla usada para
      generar el fractal.\n\
  -o, --output      Colocar la imagen de salida.\n\
Ejemplos:\n\
  tp0 -o uno.pgm\n");
      return 0;
                }

    else if (!strcmp(argv[i], "-r") || !strcmp(argv[i], "--resolution")){
                        if(!argv[i+1] || !isValidRes(argv[i+1])){
        printf("Error: valor de resolucion ingresado no valido\n");
        return -1;
      } else {
        pSeparator = strtok(argv[i+1],delRes);
        if(pSeparator != NULL){
                resWidth = atof(pSeparator);
        } else {
                exitCode = -1;
                perror("Error impreso por perror");
        }

        pSeparator = strtok (NULL,delRes);
        if(pSeparator != NULL){
                resHeight = atof(pSeparator);
        } else {
                exitCode = -1;
                perror("Error impreso por perror");
        }
                                i++;
                        }
                }

                else if (!strcmp(argv[i], "-c") || !strcmp(argv[i], "--center
                     ")){
                        if(!argv[i+1] || !isValidComplex(argv[i+1])){
                                printf("Error: valor de centro ingresado no
                                     valido\n");
        return -1;
      } else {
        char *copy = strdup(argv[i+1]);
        if(copy == NULL){
                exitCode = -1;
                perror("Error impreso por perror");
        }
                                int sign = 1;
                                if(copy[0] == '-') sign = -1;

        pSeparator = strtok(argv[i+1],delimitator);
        if(pSeparator != NULL){
                pixelPos.x = sign * atof(pSeparator);
                                        int len = strlen(pSeparator);
                                        if(sign == -1) sign = copy[len + 1]
                                             == '-' ? -1 : 1;
                                        else sign = copy[len] == '-' ? -1 :
                                             1;
        } else {
          exitCode = -1;
          perror("Error impreso por perror");
        }

        pSeparator = strtok (NULL,delimitator);
        if(pSeparator != NULL){
                pixelPos.y = sign * atof(pSeparator);
        } else {
          exitCode = -1;
          perror("Error impreso por perror");
        }
                                free(copy);
                                i++;
                }
```

```
}

else if (!strcmp(argv[i], "-w") || !strcmp(argv[i], "--width
    ")){
        if(!argv[i+1] || !isValidNumber(argv[i+1])){
                printf("Error: valor de ancho ingresado no
                    valido\n");
                return -1;
        } else {
                width = atof(argv[i+1]);
                i++;
        }
}

else if (!strcmp(argv[i], "-H") || !strcmp(argv[i], "--height
    ")){
        if(!argv[i+1] || !isValidNumber(argv[i+1])){
                printf("Error: valor de altura ingresado no
                    valido\n");
                return -1;
        } else {
                height = atof(argv[i+1]);
                i++;
        }
}

else if (!strcmp(argv[i], "-s") || !strcmp(argv[i], "--seed")
    ){
        if(!argv[i+1] || !isValidComplex(argv[i+1])){
                printf("Error: valor de seed ingresado no
                    valido\n");
                return -1;
        } else {
                char *copy = strdup(argv[i+1]);
                if(copy == NULL){
                        exitCode = -1;
                        perror("Error impreso por perror");
                }
                int sign = 1;
                if(copy[0] == '-') sign = -1;

                pSeparator = strtok(argv[i+1],delimitator);
                if(pSeparator != NULL){
                        seed.x = sign * atof(pSeparator);
                        int len = strlen(pSeparator);
                        if(sign == -1) sign = copy[len + 1]
                            == '-' ? -1 : 1;
                        else sign = copy[len] == '-' ? -1 :
                            1;
                } else {
                        exitCode = -1;
                        perror("Error impreso por perror");
                }

                pSeparator = strtok (NULL,delimitator);
                if(pSeparator != NULL){
                seed.y = sign * atof(pSeparator);
                } else {
                exitCode = -1;
                perror("Error impreso por perror");
                }
                free(copy);
                i++;
        }
}

else if (!strcmp(argv[i], "-o") || !strcmp(argv[i], "--output
    ")){
        /* open output file */
        if(!argv[i+1]){
                printf("Error: debe ingresar un archivo de
                    salida\n");
                return -1;
        } else if (!strcmp(argv[i+1], "-")){
```

```
                        continue;
                }
                else {
                        image = fopen(argv[i+1], "w");
                        if (image == NULL) {
                                if(fprintf(stderr, "No se puede abrir
                                     el archivo file %s!\n", argv[i
                                    +1]) < 0){
                exitCode = -1;
                perror("Error impreso por perror");
                                }
                                perror("Error impreso por perror");
                                return -1;
                        }
                        i++;
                }
        } else {
                if(fprintf(stderr, "Error: opcion invalida\n") < 0){
        exitCode = -1;
        perror("Error impreso por perror");
                }
                perror("Error impreso por perror");
                return -1;
        }
}

if(exitCode == 0) {
        exitCode = processImage(resWidth,resHeight,pixelPos,seed,
             width,height,image,pasoN);
}

if(exitCode == EOF){
perror("Error impreso por perror despues de procesar la imagen");
}

return exitCode;
}
```

5. Pruebas

Para las pruebas compilamos el programa con gcc de la siguiente manera:

```
$gcc main.c -o tp0
```

Luego corremos el archivo **test.sh**. Ya que las pruebas son sobre las imÃ¡genes, las vamos a realizar a ojo comparandolas con las del enunciado y con las obtenidas en un generador online (`http://usefuljs.net/fractals/`).

Cabe destacar que las imagenes del generador tienen mayor rango dinamico que las del enunciado y nosotros decidimos generarlas como en éste último.

Las imagenes obtenidas por nuestro trabajo se encuentran también en formato PNG en la subcarpeta *imagenes*. A su vez las imagenes del generador online se encuentran en *Casos de prueba*.

5.1. Caso con los valores por defecto

Se obtiene una imagen como la primera figura del enunciado:

```
$./tp0 -o uno.pgm
```

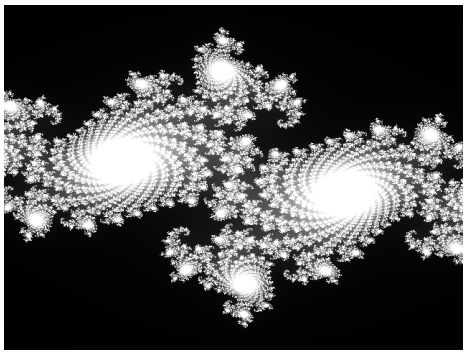

Figura 1:

5.2. Caso de imagen con zoom y otro centro

Se obtiene una imagen como la segunda figura del enunciado:

```
$ ./tp0 -c 0.282-0.007i -w 0.005 -H 0.005 -o dos.pgm
```

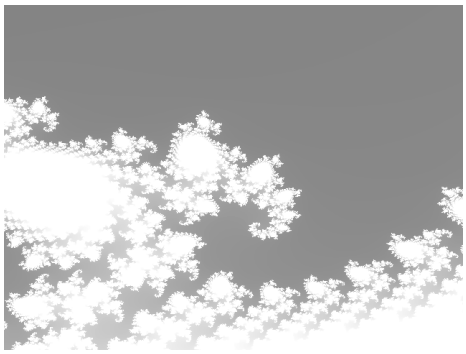

Figura 2:

5.3. Caso de imagen con ancho 1 y centro 1

Se obtiene una imagen como la primera del enunciado pero con un zoom x2 aplicado. Realizamos esta prueba por ser un caso muy facil de reproducir y comprarar con otro generador de Julia Sets.

```
$ ./tp0 -w 1 -H 1 -o tres.pgm
```

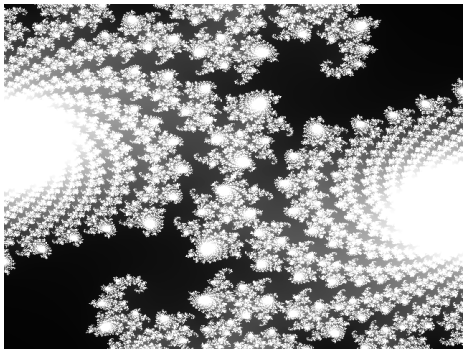

Figura 3:

5.4. Caso de imagen muy chica

Imprimimos una imagen de 8x6 para que se puedan notar claramente los pixeles en la pantalla al hacer zoom con GIMP.

```
$ ./tp0 -r 8x6 -o cuatro.pgm
```

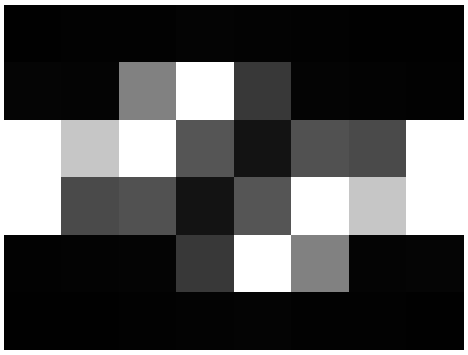

Figura 4:

5.5. Caso de imagen con otra semilla

Esta imagen usa una semilla con sus dos componentes negativas y la imaginaria mucho mas grande que la real. Esto es para comprobar que el centrado funcione correctamente, mas alla del caso del enunciado.

```
$ ./tp0 -s -0.157-1.041i -o cinco.pgm
```

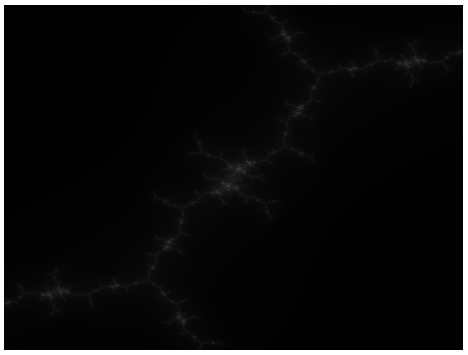

Figura 5:

5.6. Caso de imagen muy angosta

En este caso cambiamos la resolucion para obtener una imagen de un pixel de alto y 800 de ancho.
Esto se realizo para verificar que no ocurren problemas de dibujado en casos extremos. Es difícil observarla en el informe por lo que decidimos no colocarla. Sin embargo el archivo se encuentra en la misma carpeta del TP con el nombre *seis.pgm*.

```
$ ./tp0 -r 800x1 -o seis.pgm
```

6. Código S

En esta sección colocaremos el ciódigo assembly MIPS generado por NetBSD.

7. Bibliografía

1. GXemul.
   http://gavare.se/gxemul/.
2. The NetBSD project.
   http://www.netbsd.org/.
3. http://es.wikipedia.org/wiki/Conjunto_de_Julia (Wikipedia).
4. PGM format specification.
   http://netpbm.sourceforge.net/doc/pgm.html.
5. Generador de fractales.
   http://usefuljs.net/fractals/
6. GIMP.
   https://www.gimp.org/